ENHANCED BALANCING NEUMANN-NEUMANN PRECONDITIONING IN COMPUTATIONAL FLUID AND SOLID MECHANICS*

SANTIAGO BADIA†‡, ALBERTO F. MARTÍN†‡, AND JAVIER PRINCIPE†‡

Abstract. In this work, we propose an enhanced implementation of balancing Neumann-Neumann (BNN) preconditioning together with a detailed numerical comparison against the balancing domain decomposition by constraints (BDDC) preconditioner. As model problems, we consider the Poisson and linear elasticity problems. On one hand, we propose a novel way to deal with singular matrices and pseudo-inverses appearing in local solvers. It is based on a kernel identification strategy that allows us to efficiently compute the action of the pseudo-inverse via local indefinite solvers. We further show how, identifying a minimum set of degrees of freedom to be fixed, an equivalent definite system can be solved instead, even in the elastic case. On the other hand, we propose a simple modification of the preconditioned conjugate gradient (PCG) algorithm that reduces the number of Dirichlet solvers to only one per iteration, leading to similar computational cost as additive methods. After these improvements of the BNN PCG algorithm, we compare its performance against that of the BDDC preconditioners on a pair of large-scale distributed-memory platforms. The enhanced BNN method is a competitive preconditioner for three-dimensional Poisson and elasticity problems, and outperforms the BDDC method in many cases.

Key words. Domain decomposition, coarse-grid correction, balancing domain decomposition, BNN, BDDC, elasticity, parallelization, scalability.

AMS subject classifications. 65N55, 65F08, 65N30

1. Introduction. Many scientific phenomena are governed by partial differential equations (PDEs). The solution of these problems can be approximated by a discretization (and possibly linearization) of these equations e.g. via finite element (FE) methods. As a result, we end up with a linear system of equations that can be solved with the aid of computers. For complex realistic applications the systems are so large that can only be solved in distributed-memory platforms. Domain decomposition (DD) algorithms that require a partition of the original problem (domain) into sub-domains (e.g. one per processor) have been designed with this purpose. They are used as preconditioners of iterative solvers and involve the solution of local problems that can be done in parallel combined with inter-processor communication.

DD preconditioners that only require to solve local problems are highly parallel, but unfortunately they are not algorithmically scalable, i.e. the number of iterations increase with the number of processors and/or the size of the original linear system. In order to improve the situation, a coarse solver must be introduced, which couples all the sub-domains. The dimension of the coarse-grid problem is linear with respect to the number of processors, and small compared to the size of the original problem for interesting ranges of applicability. Although the amount of parallelism to be exploited

*This work has been funded by the European Research Council under the FP7 Programme Ideas through the Starting Grant No. 258443 - COMFUS: Computational Methods for Fusion Technology. A. F. Martín was also partially funded by the UPC postdoctoral grants under the programme "BKC-Atracció i Fidelització de talent al BKC". The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Red Española de Supercomputación and the Juelich Supercomputing Centre in the exploitation of the HPC for Fusion (HPC-FF) under the EFDA HPC Implementing Agreement (EFDA (08) 39/4.1).

†Centre Internacional de Mètodes Numèrics a l'Enginyeria (CIMNE), Parc Mediterrani de la Tecnologia, UPC, Esteve Terradas 5, 08860 Castelldefels, Spain ({sbadia,amartin,principe}@cimne.upc.edu).

‡Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain.

for the solution of the coarse-grid problem is small (relatively to that of the fine-grid problem), effective coarse solvers make the number of iterations almost independent of the size of the problem and number of processors.

The best suited "local" DD preconditioner is the so-called Neumann-Neumann (NN) preconditioner, since it does not require a coloring technique. Many systems arising from the discretization of PDEs do not have a zero-order (reaction) term and pure Neumann problems are singular, with a known kernel space. As a result, the solution of the local (pure) Neumann problems require a solvability condition, i.e. the residual must be balanced. Mandel proposed in [21] to consider a coarse space which makes the resulting local problems balanced, an approach related to the deflation techniques in [26, 33]. In order to attain this, the coarse and NN preconditioners must be combined in a multiplicative way. In addition, the resulting method is quasi-optimal (algorithmically scalable) with a poly-logarithmic expression of the condition number of the preconditioned system $\kappa_{\mathrm{BNN}} = 1 + \log^2(\frac{\mathtt{H}}{\mathtt{h}})$, where $\left(\frac{\mathtt{H}}{\mathtt{h}}\right)^d$ is the size of the local problems; d is the space dimension. Keeping fixed the load per processor (local system size), the number of iterations of the PCG solver is (asymptotically) independent of the number of processors and size of the global system. This desirable property is a must for achieving *weak scalability* [17].

The original BNN algorithm requires to compute the coarse-grid problem exactly and the coarse-grid system matrix has a denser stencil [23] than the original FE matrix. Further, since it is a multiplicative preconditioner, two matrix-vector products are needed per PCG iteration (see [21, 32]), i.e. two Dirichlet solvers, whereas additive DD methods only require one. These considerations have probably motivated the BDDC preconditioner by Dohrmann [9]. The BDDC preconditioner is usually considered an improved version of the BNN preconditioner [22, 23]. The basis for the coarse solver is non-conforming (when using continuous FEs) and as a result the coarse solver is not exact. Further, coarse and fine components of the preconditioner are combined in an additive way. .

2. Enhancements in this work. The target of this work is to rehabilitate the BNN preconditioner for large-scale computations in computational fluid and solid mechanics. We consider Poisson's problem and linear elasticity. The first one is the bottleneck of fluid simulations which use pressure Schur complement-type algorithms [3, 10], whereas the second is the kernel of any solid mechanics constitutive model.

The first enhancement we propose regards to the treatment of the singular local matrices. We observe that the local indefinite problems can be transformed into equivalent positive definite (PD) ones by simply fixing judiciously picked degrees of freedom. The choice of the DoFs to fix for the Laplacian problem is straightforward but less obvious in the case of three-dimensional elasticity. The final algorithm does not require to solve additional local problems or geometrical and modeling information and can straightforwardly use any off-the-shell direct solver. At every iteration of the PCG algorithm, only a backward and forward substitution is performed to evaluate the local problems, and no singular value decomposition (SVD) is used. The resulting method can also be applicable to other algorithms that involve the solution of local pure Neumann problems, as in the FETI method, and is certainly competitive with respect to previous approaches [8, 11, 13, 24]. For the (now definite) local problems, we have at our disposal the most advanced sparse direct solvers for the local problems which make an extensive use of level 3 BLAS libraries.

The second enhancement is the reduction of CPU cost per BNN-PCG iteration. We have designed an alternative algorithm which requires only one Dirichlet and one

Neumann solver per iteration. It implies a reduction of one Dirichlet solver with respect to the classical BNN preconditioned conjugate gradient [21,32]. This way, the number of local problems to be solved is identical as for additive methods like BDDC. We observe that this CPU reduction can also be attained by considering a deflated algorithm, algorithm DEF1 in [31], combined with a NN preconditioner. However, the algorithm we propose is a re-statement of A-DEF2 in [31], which has been proved to be more stable to perturbations.

Finally, we show a simple way to deal with non-balanced residual, which allows one to consider inexact solvers for the coarse problem and additive BNN preconditioners.

With the rehabilitation of the BNN algorithm proposed in this work, together with the previous observations, we carry out a detailed comparison of BNN and BDDC methods. As far as we know, there is not a comparison of this type in the literature. Again, we have considered Poisson and elasticity problems in both two and three space dimensions.

The outline of the article is as follows. In Section 3 we introduce the abstract Schwarz setting, non-overlapping DD and the BNN preconditioner. In Section 4 we show how to avoid the pseudo-inverses, make the local problems PD and consider inexact solvers. We propose a modified PCG algorithm and the corresponding elimination of one Dirichlet solver per iteration in Section 5. Section 6 includes a detailed numerical experimentation on distributed memory machines (DDMs) up to 4,096 cores and 0.4 billion DoFs, with special emphasis on the evaluation of the weak scalability. Finally, some conclusions are drawn in Section 8.

3. Abstract Schwarz theory. Let us consider a bounded polyhedral domain $\Omega \in \mathbb{R}^d$ with $d = 2, 3$. In this work, we will focus on two different second-order model problems: the Poisson problem and linear elasticity. We can state both problems as follows: find $u \in V^{\text{cont}}$ such that

$$a(u, v) = \ell(v), \qquad \text{for any } v \in V^{\text{cont}}, \tag{3.1}$$

where $a(\cdot,\cdot) : V^{\text{cont}} \times V^{\text{cont}} \to \mathbb{R}$ and $\ell(\cdot) : V^{\text{cont}} \to \mathbb{R}$. For the Poisson problem, we take $V^{\text{cont}} := H_0^1(\Omega)$ and define the bilinear form as

$$a(v_1, v_2) := \int_\Omega \nabla v_1 \cdot \nabla v_2 \mathrm{d}x, \qquad \text{for any } v_1, v_2 \in H_0^1(\Omega). \tag{3.2}$$

We can analogously state the (compressible) linear elasticity problem by considering the vector-valued space of functions, i.e. $V^{\text{cont}} := H_0^1(\Omega)^d$ and the bilinear form

$$a(v_1, v_2) := \int_\Omega \sigma_s(v_1) : \sigma_s(v_2) \mathrm{d}x, \qquad \text{for any } v_1, v_2 \in H_0^1(\Omega)^d. \tag{3.3}$$

The stress tensor is defined as $\sigma_s(v) := 2\mu\varepsilon(v) + \lambda(\nabla \cdot v)I$, where $\varepsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^t)$ is the strain rate tensor, μ and λ are the Lamé constants, I denotes the identity tensor and the super-script t stands for transpose operator. We refer to [5] for the quasi-incompressible case in the frame of BNN preconditioning.

We consider a global conforming mesh (partition) $\mathcal{T} = \{K_i : i = 1, \ldots, n_{\text{elm}}\}$ of $\overline{\Omega}$ into d-simplices, hexahedra ($d = 3$) or quadrilaterals ($d = 2$) [6]. Let us introduce the conforming FE space $V \subset H_0^1(\Omega)$. We define the finite-dimensional linear PD operator (matrix) $A : V \to V'$ by $Av = a(v, \cdot)$ for any $v \in V$ and the linear functional (vector) $f \in V'$ such that $f(v) = \ell(v)$ for any $v \in V$. Then, the discretization of

problem (3.1) leads to the following algebraic system:

$$\text{find } u \in V \text{ such that } Au = f. \tag{3.4}$$

We introduce a set of auxiliary vector spaces $\{V_i : i = 1, \dots, N\}$, a set of (at least) semi-PD operators $B_i : V_i \to V_i'$ and a set of injections $I_i : V_i \to V$. An abstract additive Schwarz preconditioner $B : V' \to B$ for A is defined by $B = \sum_{i=0}^{N} I_i B_i^{-1} I_i^t$; the transpose operator $I_i^t : V' \to V_i'$ is defined as $\langle I_i^t g, v_i \rangle = \langle g, I_i v_i \rangle$. In order to get a full rank preconditioner B, we assume that $V = \sum_{i=0}^{N} I_i V_i$. The operators $I_i B_i^{-1} I_i^t$ can alternatively be combined in a multiplicative or hybrid fashion (see [32]). In Section 3.2 we define these ingredients for the BNN preconditioner.

3.1. Non-overlapping domain decomposition. We consider a partition of Ω into subdomains $\{\Omega_i : i = 1, \dots, n_{\text{sbd}}\}$ (domain decomposition) and a partition of $\mathcal{T}$ into local meshes $\{\mathcal{T}_i : i = 1, \dots, n_{\text{sbd}}\}$ such that $\mathcal{T}_i$ is a conforming mesh of Ω_i.

Let us assume for simplicity that both the FE subdomain partition and $\mathcal{T}$ are shape regular and quasi-uniform.The diameter of Ω_i is denoted by $\mathtt{H}_i$ whereas $\mathtt{H} := \max\{\mathtt{H}_i : i = 1, \dots, n_{\text{sbd}}\}$. Analogously, given a FE $K \in \mathcal{T}$, $\mathtt{h}_K$ denotes its diameter and $\mathtt{h} = \max_{K \in \mathcal{T}} \mathtt{h}_K$.

The *interface* of Ω_i is defined by $\Gamma_i = \partial\Omega_j \setminus \partial\Omega$. The whole interface (skeleton) of the domain decomposition is $\Gamma = \bigcup_{i=1}^{n_{\text{sbd}}} \Gamma_i$. Following the notation in [7], the set of nodes of $\mathcal{T}_i$ that belong to Γ_i is denoted by Γ_h^i (idem for Γ and Γ_h). Then, we can define the space V^0 of bubble functions in V that vanish on Γ and its A-orthogonal complement H, i.e. $\langle A\varphi, v^0 \rangle = 0$ for any $\varphi \in H$ and $v^0 \in V^0$; H is usually called the space of harmonic extensions [7]. As a result, the solution of the original problem can be written as $u = u^0 + \varphi$, where

$$\langle Au^0, v^0 \rangle = \langle f, v^0 \rangle, \qquad \text{for any } v^0 \in V^0, \tag{3.5a}$$

$$\langle A\varphi, \theta \rangle = \langle f - Au^0, \theta \rangle, \qquad \text{for any } \theta \in H. \tag{3.5b}$$

Let us define the Schur complement operator $S : H \to H'$ by $\langle S\varphi, \phi \rangle = \langle A\varphi, \phi \rangle$, for any $\varphi,\ \phi \in H$. Problem (3.5b) can be re-stated as:

$$S\varphi = \gamma, \qquad \text{where } \gamma = (f - Au^0)(\theta) \quad \text{for any } \theta \in H, \tag{3.6}$$

whereas u^0 can be obtained by solving in parallel local Dirichlet problems [32]. Finally, let us remark that any function $\theta \in H$ is uniquely defined by its value on the skeleton nodes Γ_h [7, Lemma 7.5.20]. As a result, (3.6) can be stated as a linear system for the skeleton unknowns.

We construct the local FE spaces $\{V_i : i = 1, \dots, n_{\text{sbd}}\}$ associated to $\mathcal{T}_i$ (in the same way V is constructed from $\mathcal{T}$) where we enforce that any element of V_i vanishes on $\partial\Omega \cap \partial\Omega_i$. We say that Ω_i is *floating* if $\partial\Omega \cap \partial\Omega_i = \emptyset$ and *non-floating* otherwise. For every local space, we define the bilinear form

$$a_i(u, v) = \int_{\Omega_i} \nabla u \cdot \nabla v \mathrm{d}x, \qquad \text{for any } \ u,\ v \in V_i,$$

(analogously for elasticity) as well as the matrix $A_i : V_i \to V_i'$, defined by $A_i v = a(v, \cdot)$ for $v \in V_i$. A_i is semi-PD for floating sub-domains and PD for non-floating ones. Then, we define the space $H_i \subset V_i$ as the A_i-orthogonal space of bubbles V_i^0. The local Schur complement operator $S_i : H_i \to H_i'$ is defined by $\langle S_i\varphi, \theta \rangle = \langle A_i\varphi, \theta \rangle$ for any $\varphi,\ \theta \in H_i$.

3.2. Balancing Neumann-Neumann preconditioner. A DD preconditioner must be a good approximation to S^{-1} whose application allows for a maximum amount of parallelization. The key ingredients of any Schwarz preconditioner are (H_i, I_i, B_i^{-1}), i.e. the sub-spaces, injections and local preconditioners. The local spaces H_i for $i = 1, \ldots, n_{\text{sbd}}$ have been defined above. In addition, we define the coarse space H_0 in such a way that $I_i v \in H_0$ for any $v \in \ker(S_i)$ and $i = 1, \ldots, n_{\text{sbd}}$. Let us denote by $\{\phi_\alpha^i,\ \alpha = 1, \ldots, \bar{n}_{\text{ker}}\}$ a basis for the kernel of S_i without Dirichlet boundary conditions; an explicit definition of these vectors can be found in Section 4.1. For the elastic problem in three dimensions, the dimension of the potential kernel $\bar{n}_{\text{ker}}$ is six (rigid body motions) whereas only one (the constant function) for the Laplacian problem. Do not confuse $\bar{n}_{\text{ker}}$ with the true dimension of $\ker(S_i)$, which will be denoted by $n_{\text{ker}}(i)$ below. An arbitrary function $I_i\phi_\alpha^i$ will not belong to V in general, since it can violate the subdomain homogeneous Dirichlet boundary conditions. So, we consider a modification of these functions, denoted by $\text{Dir}(I_i\phi_\alpha^i)$, which consists on assigning the DoFs of Dirichlet type to zero. We consider the coarse space

$$H_0 = \{\text{Dir}(I_i\phi_\alpha^i) : i = 1, \ldots, n_{\text{sbd}},\ \alpha = 1, \ldots, \bar{n}_{\text{ker}}\}. \tag{3.7}$$

Let us remark that this particular choice of the coarse space always includes $\bar{n}_{\text{ker}}$ coarse DoFs per subdomain.

The injection operators for the local sub-spaces $I_i : H_i \to H$, for $i = 1, \ldots, n_{\text{sbd}}$ are built as follows. Since any function of H is uniquely defined by its values on Γ_h, we define

$$I_i\varphi_i(p) = \begin{cases} \frac{1}{N(p)}\varphi_i(p) & \text{if } p \in \Gamma_h^i, \\ 0 & \text{otherwise.} \end{cases},$$

where $\varphi_i(p)$ can be an scalar or a vector (e.g., for the elastic problem); $N(p)$ provides the number of subdomains that contain a given node p. Further, I_0 is the trivial injection.

Let us denote by $S_0 = I_0^t S I_0$ the Galerkin projection of the Schur complement on the coarse space H_0. We can take $B_0^{-1} = I_0 S_0^{-1} I_0^t$. In this case, the coarse preconditioner is of Galerkin type, also called exact solver [32]. On the other hand, we take $B_i^{-1} = I_i S_i^\dagger I_i^t$, using the pseudo-inverse $(S_i^\dagger)$ of the local Schur complement S_i. On non-floating subdomains $S_i^\dagger = S_i^{-1}$ and on floating ones it should be computed, in principle, from a SVD of S_i (see Section 4). When a local residual γ_i is balanced, i.e.

$$\gamma_i \in \ker(S_i)^\perp, \tag{3.8}$$

the local problem $S_i\varphi_i = \gamma_i$ has infinite solutions and $S_i^\dagger\gamma_i$ is a particular one (the one orthogonal to $\ker(S_i)$). This solvability condition, which motivated the design of the BNN preconditioner, is satisfied by $I_i^t\gamma$, when the global residual $\gamma \in H_0^\perp$. Therefore, a multiplicative combination of B_0^{-1}, the NN preconditioner

$$B_{\text{NN}}^{-1} := \sum_{i=1}^{N} I_i S_i^\dagger I_i^t \tag{3.9}$$

and B_0^{-1} is considered and the resulting hybrid preconditioner is symmetric. After some algebraic manipulations, taking into account that the coarse correction B_0 is

exact (of Galerkin type), the resulting preconditioner can be written as follows [31]:

$$B_{\mathrm{BNN}}^{-1} = B_0^{-1} + (I - B_0^{-1}S)B_{\mathrm{NN}}^{-1}(I - SB_0^{-1}). \tag{3.10}$$

The left-preconditioned BNN system reads as

$$B_{\mathrm{BNN}}^{-1}S\varphi = B_{\mathrm{BNN}}^{-1}\gamma.$$

4. Treatment of singular local matrices. One of the drawbacks associated to the BNN preconditioner (as well as FETI methods) is the requirement to deal with the possibly singular local matrices $\{S_i : i = 1, \ldots, n_{\mathrm{sbd}}\}$. Clearly, the use of SVDs for these problems [15] is possible but not acceptable in terms of CPU cost.

The treatment of the singular local matrices has been considered by many authors, mainly for the elasticity problem. De Roeck and Le Tallec proposed in [25] to perturb the problem, replacing zero pivots in a Gaussian elimination algorithm by positive (small) values (see also [13] for FETI methods). Unfortunately, this approach would prevent one to use out-of-the-box direct solvers like PARDISO [27, 28]or WSMP [18], since it is intrusive.* One popular approach is the one proposed in [11], which considers a geometrical algorithm to obtain the size of the kernel $n_{\mathrm{ker}}(i)$ and a subsequent factorization with full pivoting of the original matrix till reaching the $n_{\mathrm{ker}}(i)$ last rows, where a singular value decomposition (SVD) is used to compute the Schur complement of the remaining DoFs (see also [11, 24]). The use of full pivoting is prohibitive and the very recent work [8] proposes a clever modification of the algorithm that determines a lower bound of the maximum size of the upper diagonal block that is nonsingular and can be factorized with any existing solver whereas the Schur complement for the last rows is again performed by a SVD. A non-intrusive version of this approach requires to solve as many local problems as untouched rows in order to build the Schur complement, which has an important impact in the computational cost of the method. Even more important, this methodology does not provide a basis for the kernel, which is required for BNN preconditioning.

In this work, we consider a different approach. First, we exploit the knowledge of the differential operator in hand to propose a kernel basis detection, which is purely algebraic. Once a kernel basis is known, the action of $S_i^{\dagger}$ can be computed by solving an indefinite problem, as shown in Section 4.1. Further, the solvability condition (3.8) allows us to prove that an equivalent definite system can be obtained by simply fixing judiciously picked degrees of freedom. The final algorithm does not require to solve additional local problems or geometrical and modeling information and can straightforwardly use any off-the-shell direct solver. In Section 4.2 we present the method for the elastic problem and sketch its simplification to the Poisson problem, which is straightforward.

Finally, we observe that the BNN preconditioner with indefinite local systems does not require the solvability condition (3.8) to be a full rank and effective preconditioner. It permits one to consider inexact coarse solvers or an additive version of the BNN method.†. We finally illustrate the contributions presented herein with the example in Section 4.4.

*An intrusive approach requires the sources of the direct solver software. As an example, none of these solvers provide the sources for modifications.

†To the best of our knowledge, an additive BNN method has not been proposed so far (see Section 4.3)

4.1. Local kernel identification. Let us define the space $\mathrm{Rbd}(\Omega_i) = \{\varphi_\alpha : \alpha = 1, \ldots, 3(d-1)\}$ of potential rigid-body motions φ_α for Ω_i; this space has dimension $3(d-1)$. We consider the three-dimensional elasticity problem. The functions φ_α are defined by

$$\varphi_\alpha(p) = r_\alpha(\mathbf{x}_p), \qquad \text{for any } p \in \Omega_i,$$

where the vector-valued functions r_α are:

$$r_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \; r_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \; r_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \; r_4 = \begin{bmatrix} x_2 \\ -x_1 \\ 0 \end{bmatrix}, \; r_5 = \begin{bmatrix} -x_3 \\ 0 \\ x_1 \end{bmatrix}, \; r_6 = \begin{bmatrix} 0 \\ x_3 \\ -x_2 \end{bmatrix}.$$

The kernel of A_i (and subsequently S_i) can only be composed of linear combinations of displacement fields in the rigid-body space. The situation of floating subdomains is simple, since $\ker(S_i) \equiv \mathrm{Rbd}(\Omega_i)$. On the other hand, when three not aligned (two in two dimensions) vertices of $\partial\Omega_i$ have all the displacement components fixed, the kernel of S_i is empty. Intermediate situations cannot be straightforwardly handled.

In those intermediate situations, we propose the following algorithm, that extracts a basis for the kernel of the local problem. At every subdomain, we compute the Galerkin projection of A_i onto $\mathrm{Rbd}(\Omega_i)$, denoted by A_i^{Rbd}:

$$(A_i^{\mathrm{Rbd}})_{\alpha,\beta} = r_\alpha \cdot A_i r_\beta, \qquad \alpha, \; \beta = 1, \ldots, 6.$$

It can easily be computed by six matrix-vector products $A_i\varphi_\alpha$; no Dirichlet solvers are required at this step. Let us remark that $A_i^{\mathrm{Rbd}} \in \mathbb{R}^{6\times 6}$ only includes non-zero terms related to the imposition of boundary conditions, since all the differential terms are canceled by rigid-body motions. Now, we can perform a SVD decomposition of A_i^{Rbd}. For this very small geometrical matrix, we do not have to distinguish between very small eigenvalues that come from the ill-conditioned matrix A_i, a problem commented in [11]. Further, this step has a negligible CPU cost, due to the size of the matrix. Finally, this process does not require any additional information but the system matrix, an improvement with respect to [11]. However, some pathological situations can be devised in which the geometric matrix A_i^{Rbd} is ill-conditioned too (see some examples in [11]) even though these pathological situations are extremely unlikely after automatic mesh-partitioning of a well-posed problem. In these situations, we have to decide among those elements that belong to the coarse basis and those that do not, and a "small tolerance" must be used with this purpose. However, the great difference of BNN with respect to FETI methods is the fact that to add to the coarse space rigid body motions that are not really in the kernel but correspond to small eigenvalues is not a problem at all. On the contrary, this is the *raison d'être* of deflated algorithms, and extensively BNN preconditioning.

In these pathological situations (and finite arithmetic) the kernel of the local operator is not well defined (as it depends on a "small tolerance") and different algorithms for SVD might produce different results. It could be argued which one deserves the name BNN (the one that identifies the *real* kernel) but the present one works without any trouble (call it enhanced BNN if desired). In fact, the "small coefficient" ϵ to be picked is not a negative side of the algorithm but an strength, since we can identify ill-posed rigid-body modes and include them in the coarse space, making the selection of the coarse space adaptively based on the spectrum of the matrix, which is a strong point in the frame of deflation and projection-type iterative solvers.

We assume exact arithmetics for the subsequent discussion. We denote by $\{x_\alpha^i : \alpha = 1, \dots, n_{\ker}(i)\}$ a basis of $\ker(A_i^{\mathrm{Rbd}})$. The obvious extensions

$$v_\alpha^i = \sum_{\beta=1}^{\bar{n}_{\ker}} x_\alpha^i\big|_\beta \, r_\beta, \quad \alpha = 1, \dots, n_{\ker}(i)$$

form a basis of $\ker(A_i)$ and their restriction onto Γ_h^i, $\{\phi_\alpha^i : \alpha = 1, \dots, n_{\ker}(i)\}$, a basis of $\ker(S_i)$. At this point $S_i^\dagger \gamma$ is the solution of the problem [15]: find $\varphi \in \ker(S_i)^\perp$ such that

$$\langle S_i \varphi, \theta \rangle = \langle \gamma, \theta \rangle \quad \text{for any} \quad \theta \in \ker(S_i)^\perp, \tag{4.1}$$

which, in matrix form, reads

$$\begin{bmatrix} S_i & \Psi_i^t \\ \Psi_i & 0 \end{bmatrix} \begin{bmatrix} \varphi \\ \lambda \end{bmatrix} = \begin{bmatrix} \gamma \\ 0 \end{bmatrix},$$

where $\Psi_i \in \mathbb{R}^{n_{\ker}(i) \times n_{\mathrm{nin}}}$ with rows $\{\phi_\alpha^i\}$; n_{nin} denotes the number of interface DoFs. Note that problem (4.1) is well-posed (and it has a unique solution) and that when condition (3.8) is satisfied, $\lambda = 0$ and $S_i \varphi = \gamma$.

4.2. Definite local solvers. Our goal now is to find a system equivalent to (4.1) in which the restriction can be easily eliminated to obtain a definite matrix. To do so, let us compute a row echelon form of Ψ_i, denoted by $\tilde{\Psi}_i$ [30]. [‡] The rows of $\tilde{\Psi}_i$, that we will denote as $\tilde{\phi}_\alpha$, for $\alpha = 1, \dots, n_{\ker}(i)$, form another basis for $\ker(S_i)$. Now, on each row α we can identify the index $\beta(\alpha)$ (in fact a DoF label) such that column $\beta(\alpha)$ of $\tilde{\Psi}_i$ is e_α, for $\alpha = 1, \dots, n_{\ker}(i)$. Then we can define a $0-1$ matrix $\Delta \in \mathbb{R}^{n_{\ker}(i) \times n_{\mathrm{nin}}}$ whose rows Δ_α have only one nonzero entry $(\Delta_\alpha)_{\beta(\alpha)} = 1$, for $\alpha = 1, \dots, n_{\ker}(i)$. Abusing of notation, we will also denote by Δ the vectorial space spanned by the rows of Δ, whose dimension is also $n_{\ker}(i)$.

The Laplacian case is straightforward. For non-floating subdomains only, we pick an arbitrary $a \in \Gamma_h^i$ and define $\Delta \in \mathbb{R}(1 \times n_{\mathrm{nin}})$ such that the only nonzero value is $(\Delta)_{1a} = 1$.

We have the following result:

PROPOSITION 4.1. *Let us consider a subdomain Ω_i ($1 \le i \le n_{\mathrm{sbd}}$). When condition (3.8) is satisfied, $S_i^\dagger \gamma$ is equal to (up to an element of $\ker(S_i)$) $\bar{\varphi} \in \Delta^\perp$, solution of*

$$\langle S_i \bar{\varphi}, \theta \rangle = \langle \gamma, \theta \rangle \quad \textit{for any} \quad \theta \in \Delta^\perp. \tag{4.2}$$

Proof. First observe that $\Delta^\perp \cap \ker(S_i) = \{0\}$, that is, for any $\theta \in \ker(S_i)$:

$$(\theta, \delta) = 0 \ \text{ for any } \delta \in \Delta \quad \Longleftrightarrow \quad \theta = 0. \tag{4.3}$$

[‡] In row echelon form, there is at least one column equal to e_i, for $i = 1, \dots, n_{\ker}(i)$, where e_i is equal to one at i and zero elsewhere. As an example, for an arbitrary matrix Ψ composed by three eigenvectors, by simple row manipulations we can always end up with a matrix in row echelon form:

$$\begin{pmatrix} * & \dots & * & 1 & * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & * \\ * & \dots & * & 0 & * & \dots & * & 1 & * & \dots & * & 0 & * & \dots & * \\ * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & * & 1 & * & \dots & * \end{pmatrix}.$$

This result is easily checked by the construction of Δ. Any $\theta \in \ker(S_i)$ can be expressed as $\theta = \sum_{\alpha=1}^{n_{\ker}(i)} \theta^\alpha \tilde{\phi}_\alpha$ and any $\delta \in \Delta$ as $\delta = \sum_{\alpha=1}^{n_{\ker}(i)} \delta^\alpha \Delta_\alpha$. By construction,

$$(\tilde{\phi}_\alpha, \Delta_\beta) = 0 \qquad \text{for any } \alpha \neq \beta \tag{4.4}$$

implies $\theta^\alpha = 0$ for $\alpha = 1, \ldots, n_{\ker}(i)$.

Further, using the equality $\dim(\Delta) = \dim(\ker(S_i)) = n_{\ker}(i)$ and the orthogonality property (4.4), we readily get $H_i = \Delta^\perp \oplus \ker(S_i)$. Therefore problem (4.2) is well-posed (and has a unique solution).

Now, adding to the test function of (4.2) any element of $\ker(S_i)$, $\bar{\varphi}$ satisfies

$$\langle S_i \bar{\varphi}, \theta \rangle = \langle \gamma, \theta \rangle \qquad \text{for any } \ \theta \in H_i, \tag{4.5}$$

where we have used the fact that γ satisfies the solvability condition (3.8). In the same way ($H_i = \ker(S_i)^\perp \oplus \ker(S_i)$), adding to the test function of (4.1) any element of $\ker(S_i)$, $\varphi = S_i^\dagger \gamma$ satisfies

$$\langle S_i \varphi, \theta \rangle = \langle \gamma, \theta \rangle \qquad \text{for any } \ \theta \in H_i. \tag{4.6}$$

Subtracting (4.5) and (4.6), we readily prove that $S_i^\dagger \gamma$ and $\bar{\varphi}$ can only differ in an element of $\ker(S_i)$, proving the proposition. □

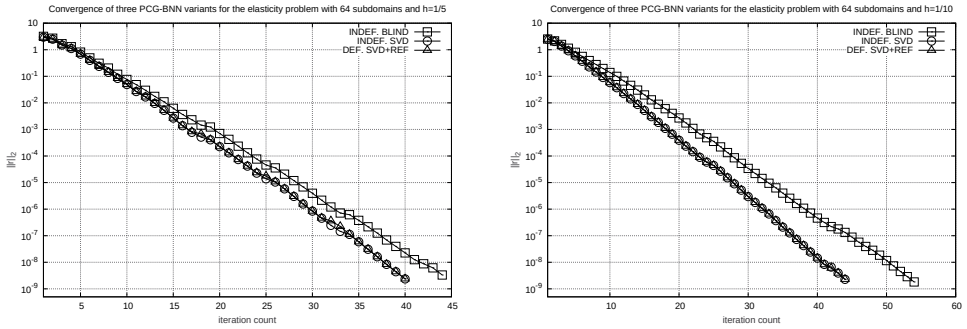
In fact, Δ is never built in practice and the system (4.2) only requires to fix the $\beta(i)$ DoF (associated to the i-component of the displacement) for $i = 1, \ldots, m$. The vectors $\tilde{\phi}_\alpha$ are not needed in a implementation of the preconditioner since the $I_i \tilde{\phi}_\alpha$ are automatically included in the coarse space H_0 defined in (3.7). Thus, we only need to know the rows $\beta(\alpha)$, which can be obtained in $\mathcal{O}(n_{\ker}^2(i))$ operations and so, in a negligible CPU cost.

This fact has an impact on the computational cost of the BNN algorithm. System (4.2) is very appealing from a computational point of view. The constraint in system (4.2) can be explicitly enforced, i.e. the corresponding Lagrange multiplier can be easily eliminated, leading to a PD matrix, without the need to introduce any perturbation to the problem. At this point, this system of equations can be solved by any direct solver for PD matrices. We show in Section 4.4 the impact of using indefinite vs. definite solvers on the CPU cost, which makes the definite version more appealing in the multiplicative case.

4.3. Inexact and additive BNN. The use of (4.2) requires $\gamma \in \ker(S_i)^\perp$. We can make it true for hybrid preconditioners where the coarse solver is solved exactly, but not for additive ones. In case the residual is not exactly balanced we can still use (4.1), which is indefinite but nonsingular. This would be, in fact, a very easy implementation of the (non-scalable) NN preconditioner. But we can also easily consider an additive BNN preconditioner

$$B_{ABNN}^{-1} = B_0^{-1} + B_{\text{NN}}^{-1},$$

where $S_i^\dagger$ is applied solving the local problem (4.1). This way, we can use any existing direct solver for indefinite matrices without modifications or perturbations; PARDISO and WSMP all have this capability. We can easily check that this is in fact the additive version of B_{BNN}^{-1}, since (4.2) and (4.1) are equivalent when applied to balanced residuals. It is easy to prove that the additive preconditioner is a full rank preconditioner. We have performed some numerical experiments so far, showing the good performance of the method. However, it is always outperformed by the enhanced version of the multiplicative BNN algorithm. For this reason, we have decided not to include these experiments in this work.



(a) $h = \frac{1}{5}$ (b) $h = \frac{1}{10}$

FIG. 4.1. *Convergence history for the 2-norm of the PCG residual of three different BNN variants.*

4.4. An illustrative example. Let us illustrate with a numerical example the effect of the techniques proposed in this section on the convergence of BNN PCG iterations. We consider the linear elasticity problem on a rectangular prism $\Omega = [0,5] \times [0,3] \times [0,1]$ with homogeneous Dirichlet boundary conditions $u = 0$ on the face contained on the $y = 0$ plane and on the corners $(5,3,1)$ and $(5,3,0)$, and $u_z = 0$ on the corners $(0,3,0)$ and $(0,3,1)$. The rest of the boundary is subject to homogeneous Neumann boundary conditions. For the discretization, $P1$ elements and two unstructured meshes of tetrahedra were generated using a mesh generator with a mesh diameter of $h = \frac{1}{5}$ and $h = \frac{1}{10}$. The resulting meshes have 18,577 tetrahedra and 3,814 nodes, and 146,268 tetrahedra and 27,348 nodes, respectively. These meshes were then partitioned into 64 subdomains (i.e., local meshes) using METIS [19].

We checked that the proposed algorithm properly identifies the kernel of the local elasticity operator for all subdomains in the partition, which for this particular case is composed of kernels with dimension 6, 5, 3, and 0. In Figure 4.1 (a) and (b) we show, for $h = \frac{1}{5}$ and $h = \frac{1}{10}$, respectively, the convergence history for the 2-norm of the PCG residual of three different BNN variants. The variant labeled as "INDEF. SVD" uses the proposed algorithm only for kernel basis detection and therefore requires the solution of local indefinite problems, that labeled as "DEF. SVD+REF" uses the proposed machinery for both basis kernel detection and subsequent identification of DoFs to obtain an equivalent symmetric-PD local problem, while the one labeled as "INDEF. BLIND" does not use this machinery at all and blindly assumes that all possible rigid-body motions are included in the kernel of the local operator. Figure 4.1 shows that a wrong basis kernel detection has a non-negligible negative impact on the convergence of the PCG solver and that this impact is more severe with finer meshes. Besides, "INDEF. SVD" and "DEF. SVD+REF" are equivalent (up-to rounding errors), as enunciated by Proposition 4.1.

5. An enhanced preconditioned conjugate gradient algorithm. The BNN preconditioner is usually applied in a PCG algorithm. As it has been observed in [31], after some manipulations (assuming exact arithmetics), the BNN-PCG algorithm can be stated in many ways. The original expression of the BNN preconditioner in (3.10) requires to compute two coarse solvers, one Neumann problem and two Dirichlet problems (associated to the application of the Schur complement system matrix S);

this *full* version is the one denoted BNN in [31], which will be denoted (f)BNN herein. However, as already pointed out by Mandel in his seminal paper [21], the application of the BNN preconditioner in the PCG algorithm always leads to a balanced residual iterate. Thus, we can always omit the first coarse preconditioner B_0^{-1} in (f)BNN by properly balancing the initial residual. That is to say, given as initial guess $\tilde{x}_0$, we compute $\tilde{r}_0 = b - S\tilde{x}_0$, the coarse correction $x_0 = \tilde{x}_0 + B_C^{-1}\tilde{r}_0$ and the balanced residual $r_0 = (I - SB_C^{-1})\tilde{r}_0$. We can easily check that the first (coarse) preconditioner can be eliminated, since $B_C^{-1}r_j = 0$ at all the iterations of the PCG algorithm. As a result, after the initial balancing, the BNN preconditioner reads as:

$$B_{BNN}^{-1} = B_C^{-1} + B_{NN}^{-1} - B_C^{-1}SB_{NN}^{-1} = B_C^{-1} + (I - B_C^{-1}S)B_{NN}^{-1}.$$

This version of the BNN preconditioner is called A-DEF2 in [31]. A further simplification is possible. Assuming again the fact that the residual is balanced, i.e. $B_C^{-1}r_j = 0$, we can simplify the previous expression as

$$B_{BNN}^{-1} = (I - B_C^{-1}S)B_{NN}^{-1}.$$

This method is the R-BNN2 preconditioner in [31], which is also the preconditioner proposed in [32, Fig. 6.2]. However, this simplification has been proved to have a negative impact in the stability and convergence properties of the resulting algorithm [31]. This is due to the fact that the *loss of balance* (due to rounding errors, inexact solvers and loose stopping criteria) for the residual in A-DEF2 is corrected at every iteration by the application of the last coarse preconditioner B_C^{-1} whereas this is not the case for R-BNN2.

A detailed comparison of these preconditioners (and many others) both theoretically and numerically can be found in [31]. The excellent numerical comparison in this reference considers the sensitivity of the convergence properties and stability of these algorithms with respect to the above mentioned perturbations. Out of this analysis, A-DEF2 proves to be the best algorithm in terms of theory and numerics. However, the way A-DEF2 is implemented in this reference requires to solve two coarse problems, two Dirichlet problems and one Neumann problem, which is prohibitive in terms of CPU cost. Let us remark that A-DEF2 method can also be expressed as follows:

$$B_{BNN}^{-1} = B_C^{-1} + B_{NN}^{-1} - B_C^{-1}SB_{NN}^{-1} = B_C^{-1}(I - SB_{NN}^{-1}) + B_{NN}^{-1}, \tag{5.1}$$

as originally proposed by Mandel [21]. This way, only one coarse solver, two Dirichlet problems and one Neumann problem are needed, as for the R-BNN2 method. This is in accordance to what is considered for a hybrid multiplicative preconditioner with exact coarse solver. As an example, we quote [32, pg. 139] when the authors refer to the BNN algorithm:

"*Therefore, there is a total of one Neumann and two Dirichlet problems on each substructure and one coarse problem in each step.*"

At this point, we can consider the distributed PCG Krylov solver, using the BNN preconditioner above. The PCG method is stated in Algorithm 1, whereas the application of the BNN preconditioner using the last expression in (5.1) can be found in Algorithm 2. In order to stress the most CPU intensive parts, we have explicitly detailed in these algorithms, the following tasks with their corresponding abbreviations:

DS: Dirichlet solve, NS: Neumann solve, CS: Coarse solve.

Algorithm 1: $x = \mathbf{PCG}(S, r_0)$

$z_0 := \texttt{i\_BNN}(r_0)$
$p_0 := z_0$
for $j = 0, \ldots,$ till convergence **do**
  $s_{j+1} = Sp_j$ (DS)
  $\alpha_j := (r_j, z_j)/(s_{j+1}, p_j)$
  $x_{j+1} := x_j + \alpha_j p_j$
  $r_{j+1} := r_j - \alpha_j s_j$
  $z_{j+1} := \texttt{i\_BNN}(r_{j+1})$
  $\beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$
  $p_{j+1} := z_{j+1} + \beta_j p_j$
end for

Algorithm 2: $z = \mathbf{i\_BNN}(r)$

$z := B_{NN}^{-1} r,$ (NS)
$s = r - Sz$ (DS)
$z := z + B_C^{-1} s$ (CS)

But there is still room for improvement. The first key observation is the following. One Dirichlet solver comes from the system matrix-vector product and cannot be eliminated. However, the second one, which comes from the residual update in a multiplicative preconditioner (the S application in Algorithm 2) can be spared. The idea is to use the fact that SI_0v_0 for $v_0 \in V_0$ can be expressed as

$$SI_0v_0 = \sum_{i=1}^{n_{\text{sbd}}} \sum_{\alpha=1}^{\bar{n}_{\text{ker}}} v_0^{(i,\alpha)} S\theta^{i,\alpha}, \tag{5.2}$$

where $\theta^{i,\alpha}$ are the elements of the H_0 basis in (3.7), i.e. $\theta^{i,\alpha} = \text{Dir}(I_i\phi_\alpha^i)$. Since all the quantities $S\theta^{i,\alpha}$ have been computed (and stored) in the assembly of the S_0 operator, one can compute SI_0v_0 using (5.2) and does not require to solve any Dirichlet solver.

Unfortunately, this fact cannot be readily used in the standard PCG Algorithm 1. It leads to the second key observation. We propose in Algorithm 3 a modified PCG algorithm, which we will prove to be equivalent (up to rounding errors) to the standard one. The idea is to consider a different treatment of the array s in an incremental way. In those places where we put $(*)$, we make use of the previous idea, in order to spare one Dirichlet solver per iteration. We have the following result:

PROPOSITION 5.1. *Algorithms 1-2 and 3-4 are equivalent.*

Proof. It is easy to see that s_1 is the same for both algorithms, since $\tilde{s}_0 = Sz_0$ and $p_0 = z_0$ at the first iteration. Next, we proceed by induction. Let us assume that s_i is equivalent for both algorithms. In Algorithm 1 $s_{i+1} = Sp_i$, whereas in Algorithm 3 we have $s_{i+1} = Sz_i + \beta_i Sp_{i-1}$, where we have used the fact that $\tilde{s}_i = Sz_i$ and the equivalence assumption at step i, i.e. $s_i = Sp_{i-1}$. We readily end the proof recalling the definition of $p_i = z_i + \beta_i p_{i-1}$. ∎

This re-statement of the BNN preconditioner makes it as expensive (in terms of Dirichlet, Neumann and coarse problems per iteration) as additive-type preconditioners like BDDC. Clearly, the enhancement of the method in Algorithms 3-4 is more

Algorithm 3: $x =$ **enh_PCG** (S, r_0)

$(z_0, \tilde{s}_0) :=$ **enh_i_BNN**(r_0)
$p_0 := z_0$
$s_0 := 0$
for $j = 0, \ldots,$ till CONV **do**
  $s_{j+1} = \tilde{s}_j + \beta s_j$
  $\alpha_j := (r_j, z_j)/(s_{j+1}, p_j)$
  $x_{j+1} := x_j + \alpha_j p_j$
  $r_{j+1} := r_j - \alpha_j s_j$
  $(z_{j+1}, \tilde{s}_{j+1}) :=$ **enh_i_BNN**(r_{j+1})
  $\beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$
  $p_{j+1} := z_{j+1} + \beta_j p_j$
end for

Algorithm 4: $(z, \tilde{s}) =$ **enh_i_BNN**(r)

$z := B_{NN}^{-1} r,$ (NS)
$\tilde{s} := Sz$ (DS)
$z_0 := B_{\mathrm{C}}^{-1}(r - \tilde{s})$
$\tilde{s} = \tilde{s} + Sz_0$ (*)
$z := z + z_0$ (CS)

important as the size of the local problems increase. In the numerical experiments section, we compare the cost of the original and enhanced formulations.

REMARK 5.1. *Let us remark that there is another way to reduce the number to Dirichlet solvers to only one by combining the deflated method in* [33] *with a NN preconditioner. The original deflated method* [26] *requires two global matrix-vector products per iteration, which for DD algorithms amounts for two Dirichlet solvers. This approach is extensively used* [1, 2, 20]. *In* [33] *the authors proposed a different deflated algorithm with enhanced stability properties, DEF-1 in* [31]*, which has a very appealing feature: the matrix-vector application related to the deflation projection acts over coarse components. Thus, this version of deflation is also superior in terms of CPU cost when using (5.2) without the need to modify the PCG algorithm. As far as we know, this fact is not clearly stated in the literature. In any case, the resulting DEF1 algorithm together with a NN preconditioner is not as stable as (f)BNN and A-DEF2 algorithms above (see* [31]*).*

6. Numerical experiments. In this section we experimentally evaluate on a pair of radically different large-scale distributed-memory machines the weak scalability of BNN and BDDC iterative sub-structuring DD methods, focusing on the rehabilitation proposed for the BNN solver.

In particular, we have carried out a detailed comparison of the enhanced BNN formulation proposed in this work with the balancing domain decomposition by constraints (BDDC) method proposed in [9], which is considered one of the most advanced scalable DD techniques. BDDC methods and the FETI-DP methods proposed in [12] have been proved to have identical condition number bounds and, under natural additional assumptions, exactly the same eigenvalues [23]. We do not introduce these techniques, for the sake of brevity, and refer the interesting reader to the nice presen-

tation in [7].

Let us just mention that BDDC methods are additive preconditioners with a non-conforming coarse space; it implies that the coarse problem is not of Galerkin type. The coarse DoFs are related to geometrical objects: corners, edges and faces (see [32, Chapter 4] for a definition). In the 2D case, corner DoFs are enough to get algorithmically scalable preconditioners whereas in 3D face DoFs must also be included.

6.1. Experimental framework. The algorithms subject of study were implemented in FEMPAR, an in-house developed OO framework which, among other features, provides the basic tools for the efficient message-passing (MPI) implementation of sub-structuring DD solvers. The MPI codes in FEMPAR heavily use standard computational kernels provided by highly-efficient vendor implementations of the BLAS. Besides, through proper interfaces to third party libraries, the local problems can be solved by sparse direct solvers. Although these kernels typically support multi-threading capabilities (OpenMP), in this work we only explore the distributed-memory (MPI) parallelism exposed by non-overlapping DD, so that the codes are executed following a pure MPI programming model with a one-to-one mapping among subdomains, MPI tasks, and computational cores of the distributed-memory machine.

All experiments reported in this section were obtained on two radically different distributed-memory platforms:

- HPC-FF: QDR Infiniband interconnected commodity cluster composed of 1080 Bull NovaScale R422-E2 blades. Each blade is equipped with two Intel Xeon X5570 QuadCore processors running at 2.93 GHz (8 computational cores in total) and 24 GBytes of DDR3 memory, and runs a full-featured SUSE SLES 11 Linux OS. The codes were compiled using Intel Fortran compiler (12.1.4) with recommended optimization flags and we used Parastation 5.0 MPI tools and libraries for native message-passing. The codes were linked against the BLAS and PARDISO [27, 28] available on the Intel MKL library (version 10.3, build 10). PARDISO is a single/multi-threaded software which follows a super-nodal approach for the efficient exploitation of the processor cache subsystem thorough the level 3 BLAS during the direct solution of large and sparse linear systems, and it has been shown to have an excellent performance relative to other parallel direct sparse solvers [16] .
- Marenostrum: Myrinet-interconnected cluster composed of 2560 JS21 compute nodes, with 4 cores (two dual-core IBM PPC970MP CPUs running at 2.3 GHz) and 8 GBytes of memory RAM each, running a full-featured SUSE 10 Linux OS. MPICH-MX port by Myricom on top of MPICH 1.2.7 was used for native message-passing, and WSMP [18], version 12.4.9, for the direct solution of sparse linear systems. WSMP is a single/multi-threaded software which follows a multifrontal aproach for the exploitation of the level 3 BLAS in the direct solution of sparse sets of linear systems. In this paper we always employ its implementation of the sparse Cholesky factorization with default values. The codes were compiled using IBM XL compilers (12.1) with recommended optimization flags, and the IBM ESSL BLAS was used for highly efficient dense linear algebra operations.

6.2. Weak scalability for 2D Poisson with structured meshes on HPC-FF. In this section we evaluate the degree of weak scalability of several sub-structuring DD solvers when applied to problem (3.2) on a rectangle $\overline{\Omega} = [0, 2] \times [0, 1]$. For the discretization, we consider a global conforming structured mesh (partition) of $\overline{\Omega}$ into

quadrilaterals and bilinear FEs (i.e., $Q1$-elements). The mesh is partitioned into rectangular grids of $4m \times 2m$ square local meshes $\mathcal{T}_{ij}$, such that they are conforming with their corresponding square subdomains Ω_{ij}, with $i = 1, \ldots, 4m$, and $j = 1, \ldots, 2m$. For the experiments, we utilize 8 cores per blade of HPC-FF (4 cores per socket) and we considered values of $m = 1, 2, 4, 6, \ldots, 22$.

Let us recall that $\frac{H}{h}$ is a measure of the local problem size; $(\frac{H}{h})^d$ is in fact the local problem size for structured mesh and partition. The number of quadrilaterals on each local mesh is therefore $\frac{H}{h} \times \frac{H}{h}$, and that of the global mesh is given by $4m\frac{H}{h} \times 2m\frac{H}{h}$. Weak scaling studies determine at which rate a given magnitude evolves while increasing the number of cores while keeping $\frac{H}{h}$ constant. For an ideal weakly scalable solver this magnitude should not degrade at all with the number of cores. In this work, we will focus on two magnitudes, the total computational time required to solve problem (3.2), and the number of iterations required to solve the preconditioned interface problem. For sub-structuring DD solvers, the total computational time is concentrated on three phases: the Schur-complement system and preconditioner set-up, and the iterative solution of the interface problem by the PCG krylov subspace solver. [§] In order to evaluate the weak scaling of the solvers under several computation/communication balances, we consider increasing values for $\frac{H}{h} = 16, 32, 64, 128, 256$ and 512, with the two extremes being the most and least communication-bounded scenarios of the sample. The largest problem size $\frac{H}{h} = 512$ was selected strategically to be the largest power-of-two that fits into the machine given a memory limit per core of 1.7 GBytes.

Figure 6.1 reports the weak scalability for the total computation time of three different implementations of the multiplicative BNN solver, while the BNN plots in Figure 6.2 illustrate that for the number of PCG iterations required to solve the interface problem. In the PCG method, we set the initial solution vector guess $x_0 = 0$, and the iteration is stopped whenever the residual r_k at a given iteration k satisfies $\|r_k\|_2 \leq 10^{-6}\|r_0\|_2$; this set-up also applies to the rest of experiments reported in this section. In the legend of Figure 6.1, `DEF` (symmetric-PD) and `IND` (symmetric INDefinite) refer to the kind of problems and the corresponding sparse direct solvers that are used for the solution of the local Neumann problems. For symmetric-PD problems, PARDISO is based on the sparse Cholesky factorization without pivoting, while for symmetric indefinite problems, it uses a more expensive sparse LDL^T factorization which, for numerical stability purposes, combines static (prior-to-factorization) pivoting via symmetric weighted matchings and classical Bunch-Kaufman dynamic (during factorization) pivoting only applied inside the supernodes [27,28]. On the other hand, `BNN` stands for the classical PCG solver (see Algorithms 1-2), while `ENH-BNN` refers to its enhanced implementation (see Algorithms 3-4).

Figure 6.1 clearly evidences that weak scaling curves for the computational time of BNN solvers result from the sum of two components, a non-scalable one that grows with the number of cores and a scalable one that is constant (or grows at a very moderate pace) with the number of cores (always for $\frac{H}{h}$ fixed). The non-scalable component includes the assembly/factorization of the coarse-grid system coefficient matrix during preconditioner set-up and the assembly of the coarse residual/solution

[§]The reader should be warned while reading this section that the total computational time is composed by phases with a different order of computational complexity with local problem size $\frac{H}{h}$. For example, the sparse direct factorization of local Dirichlet problems required for the Schur-complement system set-up grows like $\mathcal{O}((\frac{H}{h})^3)$, while each Schur complement application (i.e., forward/backward substitution) grows like $\mathcal{O}((\frac{H}{h})^2)$. We refer to [4] for a detailed exposition of the computational/memory complexity of each phase.

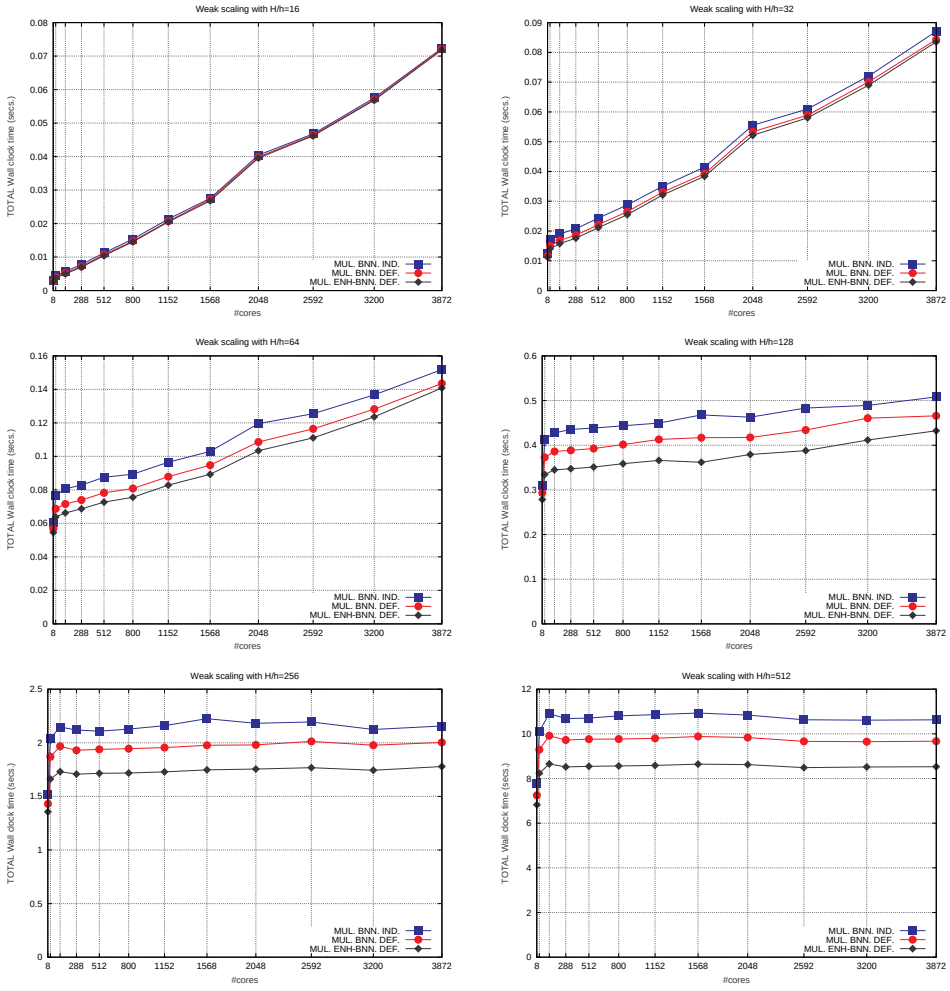


FIG. 6.1. *Weak scalability for the total computation time of three different implementations of the multiplicative BNN solver for the 2D Poisson problem on HPC-FF.*

of the coarse-system at each preconditioner application. In our current implementation, the coarse-grid system is assembled and factorized/solved *sequentially* on a prescribed processor (let us call it the root processor). This processor also has duties on the fine-grid correction level. This implementation requires the use of MPI collectives to gather the local contributions of each processor before coarse-matrix assembly/factorization and to gather the contributions of each processor to the coarse residual and scatter the coarse-grid correction before and after solving the coarse-grid system, respectively. The performance of these collectives is well-known to degrade with the number of cores. Besides, the serial solution of the coarse-grid system also results in idling overheads at each preconditioner application, as the rest of processors are blocked waiting for the root processor to complete its coarse-grid duties.

It is clearly not the intent of this section to investigate in detail at which rate each non-scalable basic building block degrades with the number of cores (a detailed discussion of these topics can be found in [4]) but to provide an overall impression

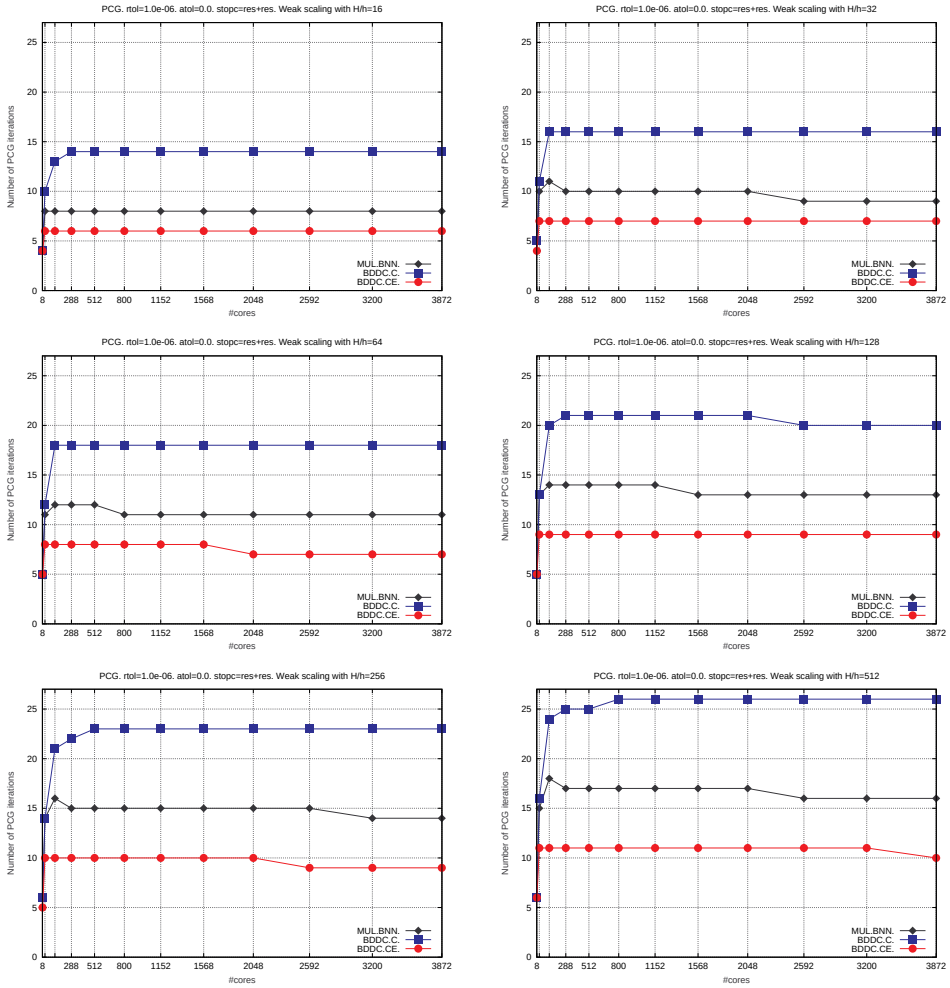


FIG. 6.2. *Weak scalability for the number of PCG iterations of the multiplicative BNN and the BDDC(c) and BDDC(ce) solvers for the 2D Poisson problem on HPC-FF.*

of what is is expected in practice from the weak scalability of these methods and the improvements proposed. On the other hand, the non-scalable component of the weak scaling curves includes the rest of computation and communication performed by the BNN solver, as e.g. the factorization/solution of local Dirichlet/Neumann problems. For "sufficiently small" $\frac{H}{h}$ (e.g., with $\frac{H}{h} = 16, 32, 64$), the non-scalable component dominates. However, the nice point is that for gradually larger $\frac{H}{h}$ the scalable part becomes more and more dominant, till the non-scalable component is completely masked, i.e. with $\frac{H}{h} = 256, 512$. This is made clear by comparing Figures 6.1 and 6.2, where the weak scalability curve for the computational time with the largest values of $\frac{H}{h}$ resembles that of the number of iterations. It is clear that the number of cores can always be increased arbitrarily so that the non-scalable component becomes dominant. However, given the memory available per core on current large-scale distributed-memory machines, and the experimental evidence we have gathered so far, this is expected to happen for simulations with several tens of thousands of cores.

Figure 6.1 also shows the benefits from using the enhancements in Sections 4-5, namely symmetric-PD solvers and to save one Dirichlet problem per PCG iteration, specially for large $\frac{H}{h}$. For example, with $\frac{H}{h} = 512$, our proposal results in a 20% maximum improvement with respect to the implementation that uses the standard BNN PCG method (it is actually a larger 35% maximum improvement if we focus on the computational time of the PCG phase). Finally, it is important to stress that the number of PCG iterations is equivalent for the three implementations, so that the proposed modifications on the PCG side do not to harm the numerical stability of the solver. The mild degradation of the number of iterations with respect to $\frac{H}{h}$ observed in Figure 6.2 is well-known from the analysis of BNN preconditioners [21, 22, 32].

Given the rehabilitation we have proposed to the BNN solver, it turns out to be interesting to answer to what extent the scalability of the BNN preconditioner is competitive with other iterative substructuring methods with coarse-grid correction available on the literature. Figures 6.3 and 6.2 compare the weak scalability for the total computation time and number of PCG iterations of the winner implementation of the multiplicative BNN solver and those of the BDDC solver with Corner (coded as `C` on the legend of the figures) and Corner+Edges (coded as `CE` on the legend of the figures) constraints [9]; here we focus on the winner implementation available in FEMPAR [4]. Hereafter, we will use the terms BDDC(c) and BDDC(ce) to refer to the latter two algorithms, respectively. For "sufficiently small $\frac{H}{h}$" (e.g., $\frac{H}{h} = 16, 32$), BDDC(ce) is the slowest method and its computational time degrades with the number of cores at the highest rate, followed by the BNN and BDDC(c), with the latter two methods very close to each other. This ranking is not surprising if one takes a closer look at the stencil of the coarse-grid coefficient matrix of each method. In particular, BDDC(c) presents the stencil corresponding to the $Q1$ FE discretization, BNN a more intricate one where neighbours of neighbours in the $Q1$ FE discretization are also connected, and finally that of BDDC(ce) resembles that of the $Q2$ FE discretization. Therefore, it is reasonable that the more intricate the stencil the higher the complexity of a sparse direct Cholesky when it is applied to solve the coarse-grid problem. This reasoning is reinforced by Table 6.1, which provides several metrics of the coarse-grid problem, namely the size and number of non-zeros in its sparse coefficient matrix, and the size of the optimal root separator of its adjacency graph. However, with gradually larger $\frac{H}{h}$, BDDC(ce) beats its competitors because its lower asymptotic number of PCG iterations (see Figure 6.2) more than pays off its higher degradation of the non-scalable component with the number of cores. We stress that the BNN method becomes highly competitive versus BDDC in terms of computational time for large $\frac{H}{h}$. This has been possible due to the techniques that we have proposed for its rehabilitation in Sections 4-5.

6.3. Weak scalability for 3D Poisson with structured meshes on Marenostrum. In this section we evaluate the weak scalability of the BNN and BDDC solvers when applied to problem (3.2) on a rectangular prism $\overline{\Omega} = [0, 2] \times [0, 2] \times [0, 1]$. For the discretization, we consider a global conforming structured mesh (partition) of $\overline{\Omega}$ into hexahedra and trilinear FEs (i.e., $Q1$-elements). The mesh is partitioned into rectangular prism grids of $2m \times 2m \times m$ cubic local meshes $\mathcal{T}_{ijk}$, such that they are conforming with their corresponding cubic subdomains Ω_{ijk}, with $i, j = 1, \ldots, 2m$ and $k = 1, \ldots, m$. For the experiments, we use 4 cores per blade of Marenostrum, with $m = 2, 3 \ldots, 10$. We consider increasing values for $\frac{H}{h} = 10, 20, 30$ and 40. The largest problem size $\frac{H}{h} = 40$ was selected strategically to be the largest multiple-of-ten that fits into the machine given a memory limit per core of 1.7 GBytes. Note that in

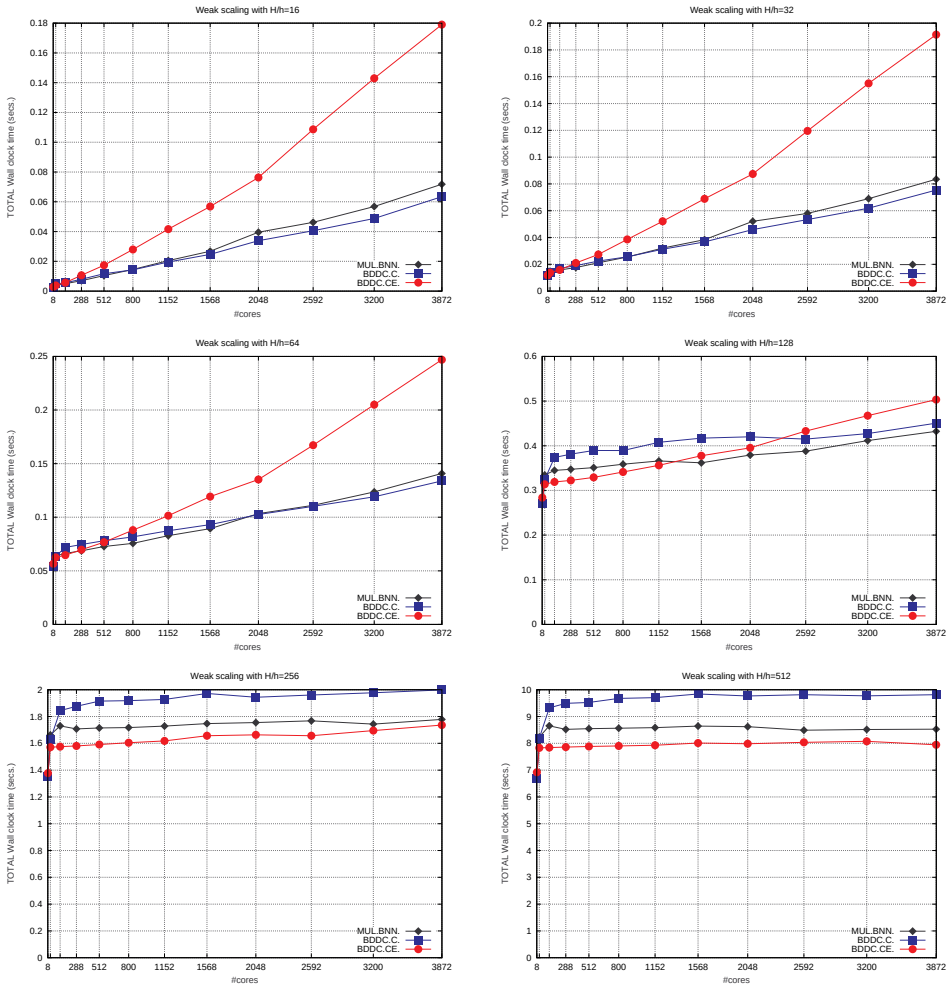


FIG. 6.3. *Weak scalability for the total computation time of the multiplicative BNN (winner implementation) and the BDDC(c) and BDDC(ce) solvers for the 2D Poisson problem on HPC-FF.*

d	metric	BDDC (c)	BDDC (ce)	BDDC (cef)	BNN
2	n_c	P	$3P$	-	P
	n_z	$9P$	$47P$	-	$25P$
	n_s	$P^{1/2}$	$2P^{1/2}$	-	$2P^{1/2}$
3	n_c	-	$4P$	$7P$	P
	n_z	-	$234P$	$462P$	$125P$
	n_s	-	$3P^{2/3}$	$4P^{2/3}$	$2P^{2/3}$

TABLE 6.1

Size (n_c), non-zeros (n_z) and optimal root separator size (n_s) for the coarse-grid scalar problem in the BDDC and BNN algorithms. A periodic structured mesh of a square ($d = 2$) or cube ($d = 3$) with $P = p^d$ subdomains is assumed, with p the number of subdomains per cartesian direction.

3D the number of hexahedra on each local cube is $\frac{\mathtt{H}}{\mathtt{h}} \times \frac{\mathtt{H}}{\mathtt{h}} \times \frac{\mathtt{H}}{\mathtt{h}}$, and that of the global mesh is given by $2m\frac{\mathtt{H}}{\mathtt{h}} \times 2m\frac{\mathtt{H}}{\mathtt{h}} \times m\frac{\mathtt{H}}{\mathtt{h}}$.

Figures 6.4 and 6.5 compare the weak scalability for the total computation time and number of PCG iterations of the winner implementation of the multiplicative BNN solver and those of the BDDC solver with Corner+Edges, and Corner+Edges+Faces (coded as `CEF` on the legend of the figures) constraints. Hereafter, we will use the term BDDC(cef) to refer to the latter algorithm. All algorithms exploit symmetric-PD solvers for the solution of the local Neumann problems. We focus on the winner implementation of the BDDC solver available in FEMPAR [4].

Figure 6.4 reveals the two components of the weak scaling curves. For sufficiently "small" $\frac{\mathtt{H}}{\mathtt{h}}$ (e.g., $\frac{\mathtt{H}}{\mathtt{h}} = 10, 20$), the total computational time is dominated by computation and communication overheads related to the solution of the coarse-grid system. To be more precise, it is dominated by the sparse Cholesky factorization of the coarse-grid coefficient matrix; this can be seen by the fact that the shape of the computational time curves resembles that of the order of arithmetic complexity of the sparse Cholesky factorization in 3D. Therefore, the rate at which the computational time of the sparse Cholesky factorization of the coarse-grid problem increases determines the rate at which the weak scalability of the methods degrade with the number of cores. BNN turns to be the quickest method and the one with the smallest rate, while BDDC(cef) the slowest and the one with the largest rate. This ranking can be justified looking at Table 6.1 with $d = 3$. Although the BNN coarse-grid sparse coefficient is denser, it is 4 and 7 times smaller than that of the BDDC(ce) and BDDC(cef), respectively, and its optimal root separator is 1.5 and 2 times smaller than that of the BDDC(ce) and BDDC(cef), respectively. The size of the optimal root separator, which can be used (actually its cube) as a lower bound for the arithmetic complexity of the sparse direct Cholesky factorization [14], accurately describes the situation observed in Figure 6.4. When $\frac{\mathtt{H}}{\mathtt{h}}$ is increased gradually, it can be observed just the opposite, i.e. BDDC(cef) is the quickest algorithm, while BNN is the slowest (although they are very close to each other). For large $\frac{\mathtt{H}}{\mathtt{h}}$, the number of PCG iterations (see Figure 6.5) determines the weak scalability of the method in terms of total computational time. A crucial observation here is that in 3D dimensions, and provided the rehabilitation that we propose for BNN, the BNN solver outperforms BDDC(ce), one of the most successful sub-structuring DD solvers available in the literature, as we increase the number of processors.

6.4. Weak scalability for 3D linear elasticity with structured meshes on HPC-FF. In this section we evaluate the weak scalability of the solvers subject of study when applied to the linear elasticity problem (3.3) on a cube $\overline{\Omega} = [0,1] \times [0,1] \times [0,1]$. For the discretization, we consider a global conforming structured mesh (partition) of $\overline{\Omega}$ into hexahedra and trilinear FEs (i.e., $Q1$-elements). The mesh is partitioned into rectangular grids of $2m \times 2m \times 2m$ cubic local meshes. For the experiments, we use all the cores in a blade of the HPC-FF, with $m = 1, \ldots, 8$. We consider increasing values for $\frac{\mathtt{H}}{\mathtt{h}} = 10, 20$ and 30. The largest problem size $\frac{\mathtt{H}}{\mathtt{h}} = 30$ was selected strategically to be the largest multiple-of-ten that fits into the machine given a memory limit per core of 1.7 GBytes.

Figures 6.6 and 6.7 compare the weak scalability for the total computation time and number of PCG iterations of the winner implementation of the multiplicative BNN solver and those of the BDDC(ce) and BDDC(cef). The winner implementation of the BNN solver is the one that, on the one hand, saves the solution of a Dirichlet problem per iteration and, on the other hand, uses the proposed machinery

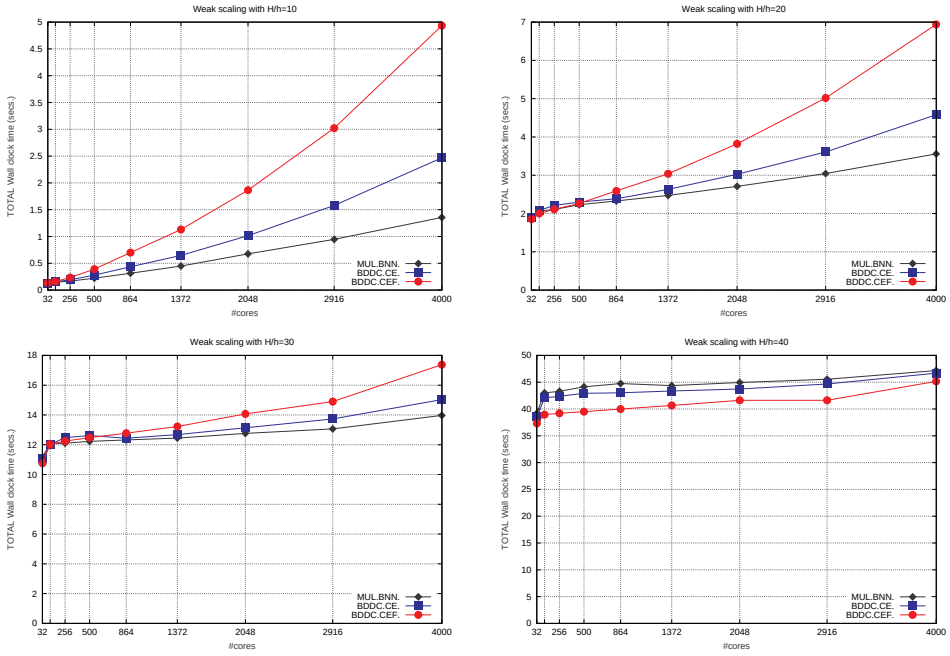


FIG. 6.4. *Weak scalability for the total computation time of the multiplicative BNN (winner implementation) and the BDDC(ce) and BDDC(cef) solvers for the 3D Poisson problem on Marenostrum.*

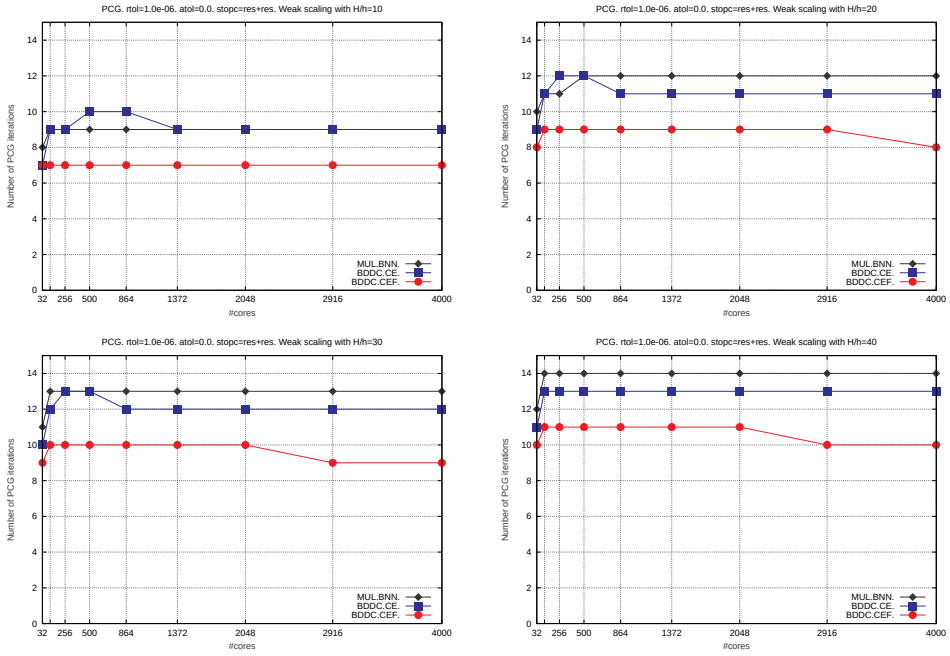


FIG. 6.5. *Weak scalability for the number of PCG iterations of the multiplicative BNN and the BDDC(ce) and BDDC(cef) solvers for the 3D Poisson problem on Marenostrum.*

for both kernel basis detection and subsequent identification of DoFs to obtain an equivalent symmetric-PD Neumann problem. As can be observed in Figure 6.6, for sufficiently "small" $\frac{H}{h}$ (e.g., $\frac{H}{h} = 10$), the overall computational time is also dominated by the sparse Cholesky factorization of the coarse-grid coefficient matrix. However, the ranking of the methods has changed with respect to that of the 3D Poisson problem (see Figure 6.4). BDDC(ce) turns to be the quickest method and the one with (by far) the slowest degradation with the number of cores, followed by the BNN and BDDC(cef), which have almost coincident performance and scalability. A characteristic of the linear elasticity problem that makes it more tough compared to the Poisson one is that the coarse-grid problem has 3 and 6 highly coupled DoFs per coarse-grid node for the BDDC and BNN methods, respectively. This leads not only to larger coarse-grid problems, but with denser sparsity patterns. As a consequence, higher memory/arithmetic complexities are expected when sparse direct solvers are used for its solution. To give some figures, n_c and n_s in Table 6.1 are multiplied by 6 and 3 for the BNN and BDDC, respectively, and n_z by 36 and 9, respectively. For $\frac{H}{h} = 30$, the scalability significantly improves but, in contrast to the 3D Poisson problem, the largest multiple-of-ten $\frac{H}{h}$ that fits into the memory of the machine is not sufficient to strike a balance among the scalable/non-scalable components such that the former dominates the overall computational time. This is because the root MPI rank, which also has fine-grid duties, has to leave (relatively to the 3D Poisson) more room to the coarse-grid problem. Anyway, the efficiency of the MPI implementation is still reasonable within this range, with a moderate 23% and 38% degradation in the computational time of the BDDC(ce) and BNN, respectively, from 512 to 4096 computational cores (i.e., global problem size multiplied by a factor of 8).

6.5. Strong scalability for 3D linear elasticity with a unstructured mesh on HPC-FF. In this section we evaluate the strong scalability of the BDDC and BNN solvers when applied to the linear elasticity problem (3.3) on a cube $\overline{\Omega} = [0,1]\times[0,1]\times[0,1]$. For the discretization, we consider a unique global conforming *unstructured* mesh (partition) of $\overline{\Omega}$ into tetrahedra and linear FEs (i.e., $P1$-elements). This mesh was generated using a mesh generation code with a mesh diameter of $h = 0.015$, which results in 2,916,260 tetrahedra and 509,064 nodes. The mesh was partitioned into 32, 64, 128, 256 and 512 subdomains using the multilevel graph partitioning algorithms available in METIS [19]. We strategically selected small loads per processor in order to reveal the effect of the serial solution of the coarse-grid problem on the strong scalability of the methods, but we stress that much better strong scalability can be attained using larger loads per processor.

Table 6.2 reports the strong scalability for the total computation time (in seconds) and number of PCG iterations for the multiplicative BNN and the BDDC(ce) and BDDC(cef) solvers; the size (n_c) and non-zeros (n_z) in the coarse-grid sparse coefficient matrix are also provided in the table. In the case of unstructured meshes and non-overlapping partitions (i.e., those provided by METIS), the BDDC method has to be supplied with a corner detection algorithm that ensures the well-posedness of the local Neumann problems and that of the global coarse-grid problem. Following the approach described in [29], our MPI implementation meets this requirement by identifying three non-colinear corners per each pair of subdomains that share a face. In general, this strategy implies the addition of more corners than those that are readily available on the underlying unstructured non-overlapping partition. Table 6.2 reveals than the BNN coarse solver is significantly lighter than that of the BDDC(ce) and BDDC(cef). While for the BNN method the number of coarse-grid

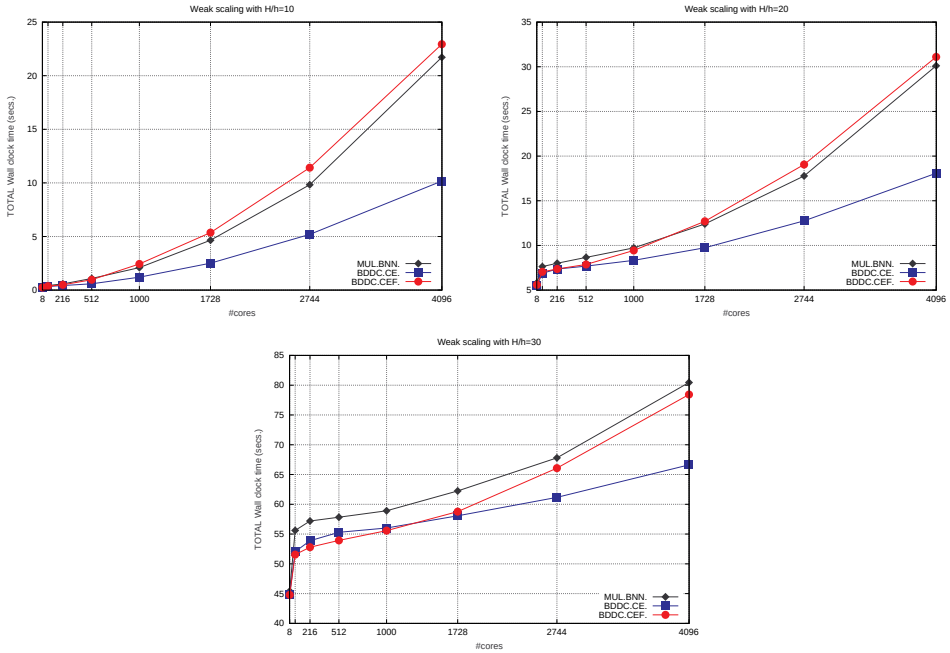


FIG. 6.6. *Weak scalability for the total computation time of the multiplicative BNN (winner implementation) and the BDDC(ce) and BDDC(cef) solvers for the 3D Elasticity problem on HPC-FF.*

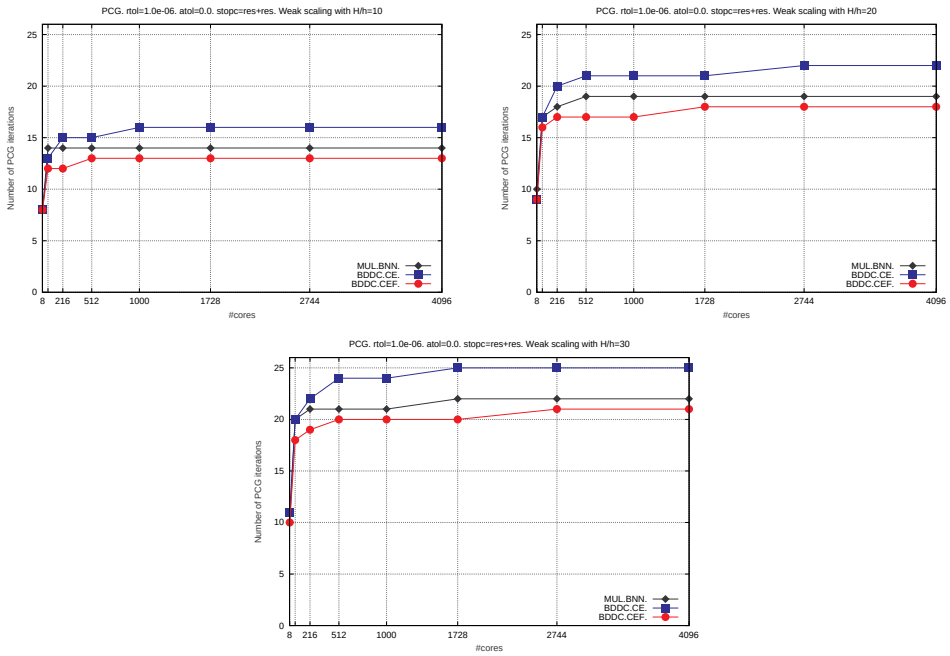


FIG. 6.7. *Weak scalability for the number of PCG iterations of the multiplicative BNN and the BDDC(ce) and BDDC(cef) solvers for the 3D Elasticity problem on HPC-FF.*

Solver		#subdomains = #cores				
		32	64	128	256	512
BNN	Time	19.4	6.87	2.66	1.48	1.35
	#iter	35	37	41	52	51
	n_c	192	384	768	1,536	3,072
	n_z	28,872	81,216	207,216	493,200	1,154,664
BDDC(ce)	Time	19.8	8.03	3.90	4.72	13.8
	#iter	25	24	24	26	25
	n_c	807	2,013	4,527	10,182	21,828
	n_z	223,263	700,209	1,758,429	4,253,436	9,602,370
BDDC(cef)	Time	20.6	8.93	5.21	8.39	24.2
	#iter	23	23	23	24	24
	n_c	1,203	2,937	6,468	14,346	30,060
	n_z	387,603	1,165,185	2,841,876	6,766,092	14,775,876

TABLE 6.2

Strong scalability for the total computation time (Time) and number of PCG iterations (#iter) for the multiplicative BNN and the BDDC(ce) and BDDC(cef) solvers for the 3D Elasticity problem on HPC-FF. The size (n_c) and non-zeros (n_z) in the coarse-grid sparse coefficient matrix are also provided.

DoFs per processor is constant independently of the geometrical properties of the underlying partition, the BDDC method is strongly dependent, and for this particular irregular underlying partition, results in much larger dimension coarse-grid subspaces. The situation is even more severe for the Elasticity problem, because additional corners have to be added to ensure the well-posedness of the preconditioner. Although this larger BDDC coarse-grid problem results in increased preconditioning robustness (#iter hardly degrades with the number of cores), the extra overhead associated with the solution of the coarse-grid problem results in a higher degradation of the strong scalability of the method. Indeed, on the range 32-512 cores, the BNN method is still effective in the reduction of the computational time with the number of cores, while that of the BDDC method even increases beyond 128 cores.

7. Implementation issues. The well-posedness of BDDC and FETI-DP preconditioners requires sets of corners (one per subdomain) such that, after enforced to zero, lead to PD local problems [9]. Unfortunately, the *geometrical* corners associated to the partition do not satisfy this in general. But this requirement is not enough. Since the coarse space is non-conforming, i.e. $H_0 \not\subset H$, the coarse solver is not a Galerkin projection of the original global problem. As a result, the *non-conforming* coarse problem can have a nontrivial kernel. In other words, the partition can induce the appearance of *mechanisms* (see [29] for a nice illustration). As a result, BDDC and FETI-DP methods require an automatic algorithm to extract enough nodes from edges and vertices to be treated as corners such that both local and coarse problems become nonsingular [9, 12, 29].

The introduction of this type of algorithm has two undesirable effects. On the one hand, the modification of the geometrical objects at the preconditioner level unquestionably complicates the implementation of the preconditioner and causes harm to software modularity. On the other hand, it increases the number of coarse DoFs and deteriorates scalability, as observed in the previous section.

Let us remark that the BNN method has a fixed number of coarse DoFs per sub-

domain (using the definition in (3.7)) and local problems are always nonsingular by its definition. Moreover, the coarse problem is of Galerkin-type and cannot be singular, due to the well-posedness of the global problem. As a result, BNN is insensitive to unstructured partitions in terms of implementation (no corner detection algorithm is required) and coarse DoFs per subdomain.

Moreover, the local BDDC problems are indefinite and the constraints related to edge or face coarse DoFs cannot easily be eliminated. In order to transform the indefinite local problems into definite ones, an explicit assembly of the Schur complement with respect to these coarse DoFs has been proposed in [9]. The implementation of this task in an acceptable way is involved and requires as many PD local solvers (of Dirichlet type) as edge and face coarse DoFs per subdomain; see also [4] for a detailed discussion on this topic. Its effect on the CPU cost of the preconditioner set-up is specially dramatic for unstructured partitions.[¶] On the contrary, the approach developed in Section 4 for the BNN method is simple and with an almost negligible associated CPU cost.

8. Conclusions. In this work, we have proposed some enhancements and extensions of the BNN preconditioner:

- A novel and very simple approach to deal with the nonsingular pure Neumann local problems, which is based on the projection of the local matrix into the potential kernel space (rigid body motions in linear elasticity) and a kernel basis extraction using a SVD. At the end of the process, it provides us with the DoFs to be fixed in the standard FE way such that we recover the same overall solution as when dealing with pseudo-inverses. This algorithm is very simple to implement, leads to PD linear systems and allows us to use out-of-the-box sparse direct solvers.
- A modification of the PCG algorithm that allows us to reduce the computational load per PCG iteration to only one Dirichlet, Neumann and coarse solver, as in additive preconditioners as BDDC, while keeping the most stable version of the preconditioner in terms of rounding error effects and perturbations, A-DEF2 in [31].
- An additive version of the BNN preconditioner which opens the door to inexact BNN algorithms.

Further, we have carried out a detailed comparison of the enhanced BNN implementation proposed herein against the very effective BDDC preconditioner initially proposed in [9]. The comparison has been performed for the Poisson and linear elasticity problems in 2D and 3D. The first problem is of special interest in computational fluid dynamics, for the so-called pressure Poisson problem, whereas the second is the core of computational solid mechanics.

As a result, the enhanced BNN preconditioner has been proved to be superior to BDDC preconditioning for large-scale computations of 3D Poisson large-scale problems (about 0.3 billion DoFs in 4,000 processors) because its associated coarse system is lighter. For the 3D linear elasticity problem, we have observed that the BDDC method outperforms the BNN preconditioner for structured meshes and partitions, with the largest simulation involving about 0.4 billion DoFs on 4,096 processors. However, the situation changes when we consider unstructured meshes and partitions. While the number of coarse functions per processor remains constant in the

[¶]This is due to the fact that the number of geometrical objects of edge and face type is larger for unstructured partitions than for structured ones.

BNN preconditioner for unstructured partitions, the BDDC preconditioner requires to introduce additional degrees of freedom in order to keep well-posed local and coarse problems, making the method less effective than BNN for real applications.

It is not the aim of this work to conclude the superiority of one preconditioner over the other one. More precisely, the aim of this comparison is to rehabilitate the BNN preconditioner for large-scale computations, demonstrating that it is (at least) as effective as BDDC (and extensively FETI-DP) preconditioning in many situations. Further, with the experience we have gained by the implementation of these preconditioners in optimized software packages, we can confidently state that the implementation of the BNN preconditioner is much simpler than the one of the other methods, since the number of coarse functions per subdomain is constant, no global and local mechanism detection is required and there is no need to modify the geometrical objects (corners, edges and faces) in order to make the local and coarse solvers well-posed (see [9, 12, 29]). Further, the approach proposed in this work to end up with PD local problems is also simpler and less CPU intensive than the one used for BDDC preconditioners in [9] (see also [4]).

References.

[1] R. Aubry, F. Mut, S. Dey, and R. Löhner, *Deflated preconditioned conjugate gradient solvers for linear elasticity*, International Journal for Numerical Methods in Engineering **88** (2011), no. 11, 1112–1127.

[2] R. Aubry, F. Mut, R. Löhner, and J. R. Cebral, *Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation*, Journal of Computational Physics **227** (2008), no. 24, 10196–10208.

[3] S. Badia and R. Codina, *Algebraic pressure segregation methods for the incompressible navier-stokes equations*, Archives of Computational Methods in Engineering **15** (2007), 1–52. 10.1007/BF03024946.

[4] S. Badia, A. F. Martín, and J. Príncipe, *Implementation and scalability analysis of balancing domain decomposition methods*, Submitted (2012).

[5] L. Beirão da Veiga, C. Lovadina, and L. Pavarino, *Positive definite balancing Neumann–Neumann preconditioners for nearly incompressible elasticity*, Numerische Mathematik **104** (2006), no. 3, 271–296.

[6] D. Braess, *Finite elements: Theory, fast solvers, and applications in solid mechanics*, Cambridge University Press, 2007.

[7] S. C. Brenner and R. Scott, *The mathematical theory of finite element methods*, 3rd edition, Springer, 2010.

[8] T. Brzobohatý, Z. Dostál, T. Kozubek, P. Kovář, and A. Markopoulos, *Cholesky decomposition with fixing nodes to stable computation of a generalized inverse of the stiffness matrix of a floating structure*, International Journal for Numerical Methods in Engineering **88** (2011), no. 5, 493–509.

[9] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM Journal on Scientific Computing **25** (2003), no. 1, 246–258.

[10] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*, Oxford University Press, 2005.

[11] C. Farhat and M. Géradin, *On the general solution by a direct method of a large-scale singular system of linear equations: application to the analysis of floating structures*, International Journal for Numerical Methods in Engineering **41** (1998), no. 4, 675–696.

[12] C. Farhat, K. Pierson, and M. Lesoinne, *The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems*, Computer Methods in Applied Mechanics and Engineering **184** (2000), no. 2–4, 333–374.

[13] C. Farhat and F.-X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, International Journal for Numerical Methods in Engineering **32** (1991), no. 6, 1205–1227.

[14] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis **10** (1973), no. 2, 345–363.

[15] G. H. Golub and C. F. V. Loan, *Matrix computations*, JHU Press, 1996.

[16] N. I. M. Gould, J. A. Scott, and Y. Hu, *A numerical evaluation of sparse direct solvers for the*

solution of large sparse symmetric linear systems of equations, ACM Trans. Math. Softw. **33** (2007), no. 2.
[17] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to parallel computing*, 2nd ed., Addison-Wesley, 2003.
[18] A. Gupta, *WSMP: Watson sparse matrix package (Part-I: direct solution of symmetric sparse systems)*, Technical Report RC 21886, IBM T. J. Watson Research Center, Yorktown Heights, NY, 2000.
[19] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput. **20** (1998), no. 1, 359–392.
[20] R. Löhner, F. Mut, J. R. Cebral, R. Aubry, and G. Houzeaux, *Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation: Extensions and improvements*, International Journal for Numerical Methods in Engineering **87** (2011), no. 1-5, 2–14.
[21] J. Mandel, *Balancing domain decomposition*, Communications in Numerical Methods in Engineering **9** (1993), no. 3, 233–241.
[22] J. Mandel and C. R. Dohrmann, *Convergence of a balancing domain decomposition by constraints and energy minimization*, Numerical Linear Algebra with Applications **10** (2003), no. 7, 639–659.
[23] J. Mandel, C. R. Dohrmann, and R. Tezaur, *An algebraic theory for primal and dual substructuring methods by constraints*, Applied Numerical Mathematics **54** (2005), no. 2, 167–193.
[24] M. Papadrakakis and Y. Fragakis, *An integrated geometric–algebraic method for solving semi-definite problems in structural mechanics*, Computer Methods in Applied Mechanics and Engineering **190** (2001), no. 49–50, 6513–6532.
[25] Y. H. De Roeck and P. Le Tallec, *Analysis and test of a local domain decomposition preconditioner*, Fourth international symposium on domain decomposition methods for partial differential equations, 1991, pp. 112–128.
[26] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, *A deflated version of the conjugate gradient algorithm*, SIAM Journal on Scientific Computing **21** (2000), no. 5, 1909–1926.
[27] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems **20** (2004), no. 3, 475–487.
[28] ———, *On fast factorization pivoting methods for sparse symmetric indefinite systems.*, ETNA. Electronic Transactions on Numerical Analysis [electronic only] **23** (2006), 158–179.
[29] J. Šístek, M. Čertíková, P. Burda, and J. Novotný, *Face-based selection of corners in 3D substructuring*, Mathematics and Computers in Simulation **82** (2012), no. 10, 1799–1811.
[30] G. Strang, *Linear algebra and its applications*, Thomson, Brooks/Cole, 2006.
[31] J. Tang, R. Nabben, C. Vuik, and Y. Erlangga, *Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods*, Journal of Scientific Computing **39** (2009), no. 3, 340–370.
[32] A. Toselli and O. Widlund, *Domain decomposition methods*, 1st ed., Springer, 2004.
[33] C Vuik, A Segal, and J.A Meijerink, *An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients*, Journal of Computational Physics **152** (1999), no. 1, 385–403.